Introduction to Data Mining

CS3753 Data Science

Prof. Weining Zhang

Topics

- What is a Data Mining
- Python Scikit-Learn package
- Classification
- Train and use a decision tree
- Classification accuracy
- Cross Validation and ROC

What is a Data Mining

Data Mining is a process to discover useful knowledge from data, and involves a number of steps:

- Collecting relevant data
- Data preprocessing, include EDA, cleaning, reduction, noise removal, filling missing values, normalization, discretization, transformation, etc.
- Apply DM algorithms to learn patterns
  - Classification: decision tree, neutral network, SVM, etc.
  - Cluster analysis
  - Outlier detection
  - Association analysis
  - etc.
- Presentation of learned knowledge, visualization, etc.
- Application of learned models

Python has tools to perform many data mining tasks.

Python Scikit-Learn Package

The sklearn (http://scikit-learn.org/stable/modules/classes.html) is a Machine Learning package in Python, providing tools for data mining and data analysis and is built on NumPy, SciPy, and matplotlib

Classification

Identifying to which category an object belongs to.

- Applications: Spam detection, Image recognition.
- Algorithms: SVM, nearest neighbors, random forest, …

Regression

Predicting a continuous-valued attribute associated with an object.

- Applications: Drug response, Stock prices.
- Algorithms: SVR, ridge regression, Lasso, …

Clustering

Automatic grouping of similar objects into sets.

- Applications: Customer segmentation, Grouping experiment outcomes
- Algorithms: k-Means, spectral clustering, mean-shift, …

Dimensionality reduction

Reducing the number of random variables to consider.

- Applications: Visualization, Increased efficiency
- Algorithms: PCA, feature selection, non-negative matrix factorization.

Model selection

Comparing, validating and choosing parameters and models.

- Goal: Improved accuracy via parameter tuning
- Modules: grid search, cross validation, metrics.

Preprocessing

Feature extraction and normalization.

- Application: Transforming input data such as text for use with machine learning algorithms.
- Modules: preprocessing, feature extraction.

In [ ]:
```
%matplotlib inline

from __future__ import division
from numpy.random import randn
import numpy as np
from numpy import linalg as LA
import os
import matplotlib.pyplot as plt
np.random.seed(12345)
plt.rc('figure', figsize=(10, 6))
from pandas import Series, DataFrame
import pandas as pd
import statsmodels.api as sm
from scipy import stats
np.set_printoptions(precision=4, threshold=500)
pd.options.display.max_rows = 100
import graphviz
```

Classification

- Train a predictive model from a set of training data
  - Each data has a class label
  - A learner algorithm fits the data into a model
  - There are various models
    - Decision trees
    - K-Nearest Neighbor
    - Naive Bayes
    - Support Vector Machine
- Use the trained model to predict class label for new data that the learner has never seen before

Use sklearn Predictive Models

- sklearn comes with a number of predictive model learning algorithms, specifically, a decision tree learnining algorithm.
- The basic use pattern is to
  - Create a model
  - fit the model using a dataset
  - use the model to predict one or more data items

Example: Decision Tree Learning

- Load a dataset iris, which comes with the sklearn package
  - The data set contains 150 records under 5 attributes: Petal Length, Petal Width, Sepal Length, Sepal width and Class.
  - There are 50 samples for each of three species of Iris flower (Iris setosa, Iris virginica and Iris versicolor).

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

```
In [ ]: iris = load_iris()
        data1 = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                             columns= iris['feature_names'] + ['target'])
        data1
```

- Instantiate the DecisionTreeClassifier predictive model

```
In [ ]: clf = tree.DecisionTreeClassifier(random_state=0)
```

- Fit the model to the data (or learn the model using the training data)

```
In [ ]: cols = iris.feature_names
        clf = clf.fit(data1[cols], data1.target)
```

- Visualize the learned decision tree model

  Here, we use the graphviz package. You may need to install it.

```
In [ ]: dot_data = tree.export_graphviz(clf, out_file=None,
                                  feature_names=cols,
                                  class_names='status',
                                  filled=True, rounded=True,
                                  special_characters=True)
        graph = graphviz.Source(dot_data)
        graph
```

Use the Decission Tree to Predict Unseen Data

- The input to the predict() function needs to be a two dimensional array, representing a list or a set of tuples.
- The output is a 1-dimensional array with one prediced class for each input tuple

```
In [ ]: t = np.array([[5.0, 2.9, 1.5, 0.1]])
        t
```

```
In [ ]: clf.predict(t)
```

Measure Classification Accuracy

- Seperate data into training set and testing set
- Train model using training set and measure classification accuracy using testing set
- Overfit a model with the training set will reduce accuracy on unseen data
- Accuracy is affected by the learner algorithm, training set and the testing set

Cross Validation

k-fold cross validation is a process to measure the classification accuracy of the learned model.

- The data set is randomly divided into k equal-size units.
- The process contains k iterations. In iteration i, the unit i is used to test the accuricy of the model, and the remaining $k-1$ units are used together to learn the model.
- The k accuricy is used to describe the quality of the learned model

```
In [ ]: cross_val_score(clf, iris.data, iris.target, cv=10)
```

Confusion Matrix

- Keep counts of testing data
- For all pairs of true and predicted classes
  - number of correctly predicted the class labels
  - number of incorrectly predicted the class labels

```
In [ ]: def confusionMatrix(actual, pred):
            classes = np.unique([actual, pred])
            cm = [[sum((actual == i) & (pred == j))
                for i in classes]
                    for j in classes]
            cm = pd.DataFrame(cm, index=classes, columns=classes)
            cm.index.name='actual'
            cm.columns.name='pred'
            return cm
```

```
In [ ]: import sklearn.linear_model as lm
        lr = lm.LogisticRegression()
        x = data1[cols]
        y = data1.target
        lr.fit(x, y)
        pred_prob=lr.predict_proba(x)
        confusionMatrix(y, pred_prob[:,1] >= 0.5)
```

```
In [ ]: import sklearn.model_selection as ms
        import sklearn.metrics as metrics

        pred_prob=ms.cross_val_predict(lr, x, y, method='predict_proba')
        confusionMatrix(y, pred_prob[:,1] >= 0.5)
```

```
In [ ]: metrics.accuracy_score(y, pred_prob[:,1]>=0.5)
```

```
In [ ]: metrics.cohen_kappa_score(y, pred_prob[:,1]>=0.5)
```

Receiver Operating Character (ROC)

- Shows True Positive Rate (TPR) vs False Positive Rate (FPR)
- Simple method of getting a ROC curve using cross-validation:
  - Collect probabilities for instances in test folds
  - Sort instances according to probabilities
- The next example generates an ROC curve for each fold and averages them

Example: Cross-Validate with ROC

In [ ]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import StratifiedKFold
from sklearn import svm
from scipy import interp
```

Prepare Data From IRIS Data

- Make sure there are only two class labels
- Extend from 4 columns to 804 columns with random values in new columns

In [ ]:
```
X = iris.data
y = iris.target
X, y = X[y != 2], y[y != 2]
n_samples, n_features = X.shape

# Add 800 noisy features
random_state = np.random.RandomState(0)
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]
```

Cross Validation and ROC

- Do 6-folds cross validation
- In each iteration,
  - train an SVM classifier,
  - classify the test set,
  - compute TPR and FPR
  - Plot a ROC
- Compute and draw the average ROC and the upper/lower margins

In [ ]:

```
# ##############################################################################
# Classification and ROC analysis

# Run classifier with cross-validation and plot ROC curves
cv = StratifiedKFold(n_splits=6)
classifier = svm.SVC(kernel='linear', probability=True,
                     random_state=random_state)

tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

i = 0
for train, test in cv.split(X, y):
    probas_ = classifier.fit(X[train], y[train]).predict_proba(X[test])
    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(y[test], probas_[:, 1])
    tprs.append(interp(mean_fpr, fpr, tpr))
    tprs[-1][0] = 0.0
    roc_auc = auc(fpr, tpr)
    aucs.append(roc_auc)
    plt.plot(fpr, tpr, lw=1, alpha=0.3,
             label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc))

    i += 1
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
         label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
plt.plot(mean_fpr, mean_tpr, color='b',
         label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
         lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                 label=r'$\pm$ 1 std. dev.')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```